Software Development Principles

Lecture 4
Definitions 2

Lecturer:

Karen Nolan

karen.nolan@it-tallaght.ie

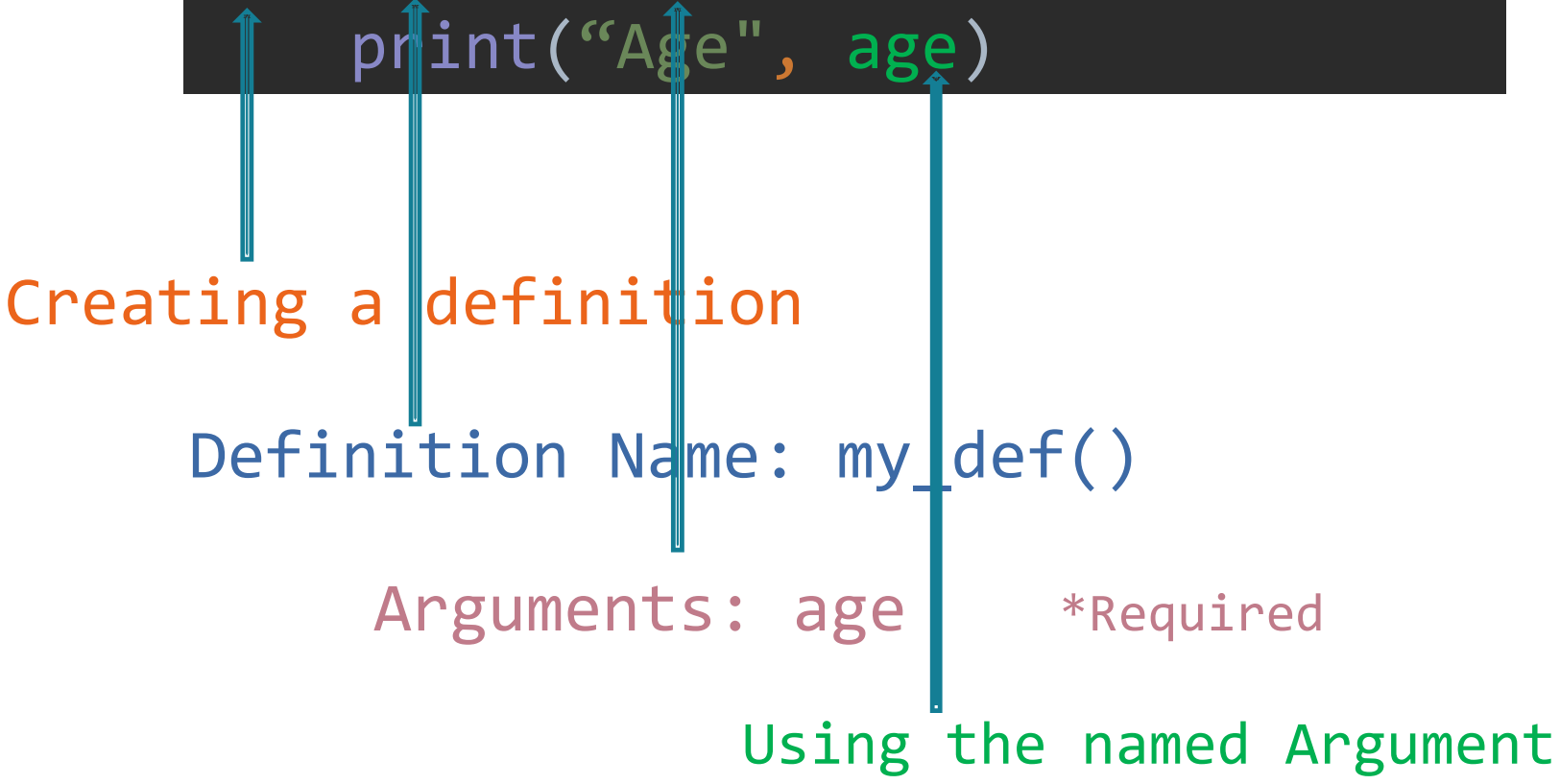
Topics

- Definition Arguments Part 2
  - Single and Multiple Args (Requirement)
  - Keyword Args
  - Default Args
  - Pass by value and Variable Scope
- Definitions with Returns
  - Return Values
  - Return Types
- Definitions calling Definitions
  - Brief Introduction

Python Definitions Args

- Recap:

```
def my_def(age):
    print("Age", age)
```



Python Definitions Args

- Recap:

```
def my_def(age):
    print("Age", age)
```

- What would happen if we called:

```
my_def()
```

Python Definitions Args

```
def my_def(age):
    print(“Age", age)
```

```
my_def()
```

```
Traceback (most recent call last):
  File "C:/Users/webar_000/Desktop/Temp/Python/SDEV2/mainTest.py", line
20, in <module>
    my_def()
TypeError: my_def() missing 1 required positional argument: 'age'
```

Python Definitions Args

- Multiple Args:

```
def my_def2(name, age):
    if age < 17:
        print(name, "you are too young to drive")
    else:
        print(name, "you are old enough to drive")
```

- What would happen with the following:

```
my_def2(21, “karen")
```

Python Definitions Args

- Multiple Args:

```
def my_def2(name, age):
    if age < 17:
        print(name, "you are too young to drive")
    else:
        print(name, "you are old enough to drive")
```

- What would happen with the following:

```
my_def2(21, “karen")
```

TypeError: unorderable types: str() < int()

Python Definitions Keyword Args

- Keyword Arguments:

```
def my_def2(name, age):
    if age < 17:
        print(name, "you are too young to drive")
    else:
        print(name, "you are old enough to drive")
```

- The definition keeps the same structure

- But the code calling the definition gets modified, this allows Arguments to be provided to the definition, unordered (this also has additional uses later on).

Python Definitions Keyword Args

- Multiple Args:

```
def my_def2(name, age):
    if age < 17:
        print(name, "you are too young to drive")
    else:
        print(name, "you are old enough to drive")
```

- Definition Arguments using Keywords:

```
my_def2(age=21, name="karen")
```

- This allows Python to match values with parameters.

Python Definition Overloading

- Definitions can have multiple purposes.
- Sometimes we require similar or identical code, but the Arguments can be different.
- Lets first look at a basic Default Argument:

Python Definition Overloading

```
def print_my_age(age=None):
    if age == None:
        print("You are too old to say!")
    else:
        print("Your age is", age)
```

- This option age=None is a very important feature called Default Arguments.
- This assigns no value if no Args are entered, which can also be tested.

Python Definition Overloading

- This option age=None is a very important feature called Default Arguments.

```
def print_my_age(age=None):
    if age == None:
        print("You are too old to say!")
    else:
        print("Your age is", age)
```

```
1) print_my_age()
2) print_my_age(35)
```

- What would you expect to see printed?

Python Definition Overloading

- The test can also consist of

```
def print_my_age(age=None):
    if age is not None:
        print(Your age is", age)
    else:
        print("You are too old to say!")
```

OR

```
def print_my_age(age=None):
    if age is None:
        print("You are too old to say!")
    else:
        print("Your age is", age)
```

Python Definition Overloading

- The default argument can also assign a specific value other than None

```
def print_my_age(age=18):
    print("Your age is", age)
```

- This will assign 18 to age, if no value is entered:

```
print_my_age()
print_my_age(35)
```

Output:

Your age is 18
Your age is 35

Python Definition Overloading

- Using default arguments you can define a function in such a way that there are multiple ways to call it.
- Depending on the function definition, it can be called with zero, one, two or more parameters.
- This is known as **method overloading**.
- To clarify method overloading, we can now call the function **print_my_age** in two ways.

Python Definition Class Example 1

- Create a definition that has three arguments for compound interest
- The formula for compound interest is:

$$\text{AmountAfterInterest} = \text{PrincipalAmount}(1 + \text{interestAsADecimal})^{\text{Years}}$$

- The definition must take in principal amount and interest as a ***percentage***
- The years defaults to 1 if no year is entered

Python Definition Solution 1

```
def compound_interest(principal_amount, interest_rate, years=1):
    amount = principal_amount * ((1 + (interest_rate / 100)) * years)
    amount = round(amount, 2)
    print* ("Total amount: ", amount)
```

Python Definition Class Example 2

- Definitions can have multiple purposes..

```
def largest(number1, number2):
    if number1 > number2:
        print(number1)
    elif number1 < number2:
        print(number2)
    else:
        print("Numbers are equal")
```

- Modify the above definition to allow for the user to either enter 2 numbers (to see which is larger), or enter two strings (to see which is longer):

```
largest(“Karen", "Mary")
```

Python Definition Solution 2

```
def largest(number1=None, number2=None, name1=None, name2=None):
    if number1 is not None and number2 is not None:
        if number1 > number2:
            print(number1)
        elif number1 < number2:
            print(number2)
        else:
            print("Numbers are equal")
    if name1 is not None and name2 is not None:
        if len(name1) > len(name2):
            print(name1)
        elif len(name2) > len(name1):
            print(name2)
        else:
            print("Names length are equal")
```

Python Definition Solution 2

```
largest(number1=5, number2=7)
largest(name1=“Karen", name2="Darren")
```

```
Output:
      7
      Darren
```

Python Definition Scope

- Scope is a concept in many programming languages, Python is no different.
- Scope essentially means what variables can be seen where.
- This leads onto pass by value and pass by reference

  Python only uses pass by reference
- Let have a look at three specific scenarios

Python Definition Scope Scenario 1

- Can a definition access a variable from the code that calls it?

```
def print_number():
    print(myNumber)

myNumber = 7

print_number()

myNumber += 1

print_number()
```

Python Definition Scope Scenario 2

- Can the main code access variables created in a definition?

```
def print_number():
    x = 7
    print(x)


print_number()

x += 1

print_number()
```

Python Definition Scope Scenario 3

- If an Argument is modified, does it modify the variable from which it was set from

```
def print_number(number_in):
    number_in += 1
    print("printed in the definition:", number_in)

x = 7
print("printed before definition:", x)
print_number(x)
print("printed after definition:", x)
```

Python Definition Scope

- The below is an overview of Python Scope with definitions:

```
x = 7                    # Global Variable


def print_number():
    print(x)             # Prints Global Variable
    y = 9                # Local Variable
    print(y)             # Prints Local Variable
```

- Y is only visible inside of the definition body
- X is a global variable visible throughout the program body

Python Definition Scope (Shadowing)

- The below is an example or variable shadowing using Arguments:

```
x = 7                   # Global Variable

def print_number(x): # Local Variable
    x += 1
    print(x)

print_number(x)
print(x)
```

- x is recreated as a local variable inside of the definition.
- Thus modifications are only on the local variable x, not the global variable x

Python Definition Scope (Shadowing)

```
x = 7                    # Global Variable

def print_number(x):
    x += 1
    print(x)

print_number(x)
print(x)
```

- This means that a local copy is created. (two variables called x)

Python Definition Pass by Reference

- Recap:

```
x = 7
y = x
y += 1

print(x)
print(y)
```

- What will be printed out?
-

Python Definition Pass by Reference

- Recap:

```
x = [1, 2, 3]
y = x
y[0] = 9

print(x)
print(y)
```

- What will be printed out?

Python Definition Pass by Reference

- Recap: Shadow copy and deep copy

```
myList = [1, 2, 3, 4]                    # Global Variable


def print_number(myList):
    myList[3] = 33
    print(myList)

print_number(myList)
print(myList)
```

- Shadow copy and deep copy with lists, is based on how variables are stored.
- Objects like lists, pass reference values and not the contents
- What will be printed out from the above code?

Python Definition Pass by Reference

```
myList = [1, 2, 3, 4]                    # Global Variable

def print_number(myList):
    myList[3] = 33
    print(myList)

print_number(myList)
print(myList)
```

- Python only uses pass by reference
- Be cautious with advanced data structures like Lists

Python Definition Pass by Reference

- Local Scope of myList

```
myList = [1, 2, 3, 4]                    # Global Variable


def print_number(myList):
    myLocalList = myList[:]              # Deep copy to local variable
    print(myLocalList)
    myLocalList[0] = 99
    print(myLocalList)

print_number(myList)
print(myList)                          # myLocalList not accessible
```

Python Definition Returns

- Up to now, each definition prints out the results.
- This is not always desired.
- Think about, if this was a webpage, would this suit html?
- You might not always want to print the same text.

```
def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    print("The volume of the sphere is", volume)
```

Python Definition Returns

- The below method is a very useful method.

- But lets say we want to use it to calculate weight of a planet?
  - The density of earth is 5,515.3 kg / m^3
  - The radius of the earth is 6371000 m

```
def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    print("The volume of the sphere is", volume)
```

Python Definition Returns

- We can use returns to send information back to code which called the definition.

```
def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    return volume
```

Python Definition Returns

- The density of earth is 5,515.3 kg / m^3
- The radius of the earth is 6371000 m

```
def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    return volume
```

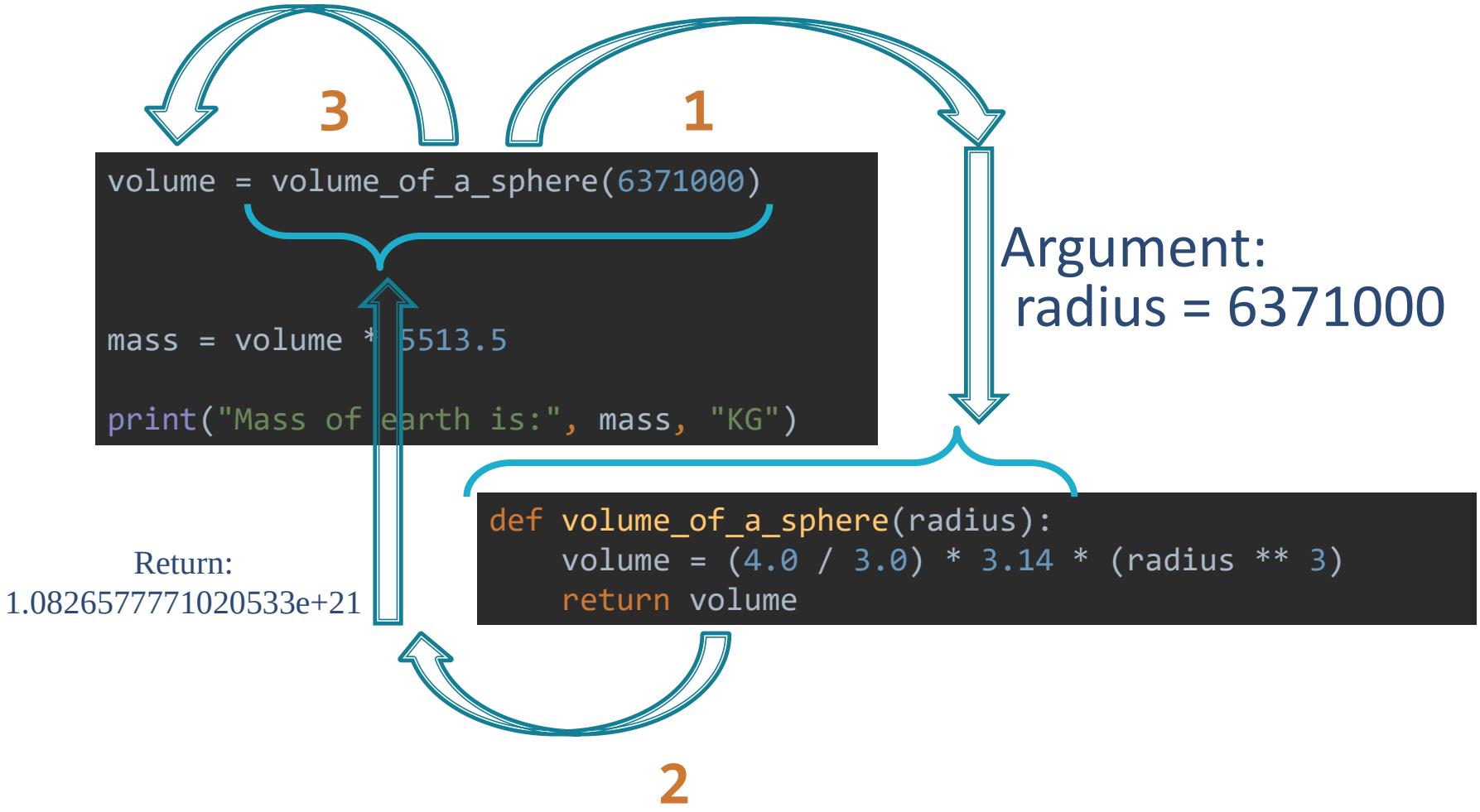
```
volume = volume_of_a_sphere(6371000)

mass = volume * 5513.5

print("Mass of earth is:", mass, "KG")
```

Output:

Mass of earth is: 5.969233654052l7e+24 KG

Python Definition Returns

Stores returned data in volume.



Python Definition Example 3

- Modify the below definition, to return the compound interest to the program body:
  - The years is 5
  - The principle amount is taken in from the user
  - The interest rate is 3.23%

```
def compound_interest(principal_amount, interest_rate, years=1):
    amount = principal_amount * ((1 + (interest_rate / 100)) ** years)
    print("Total amount: ", amount)
```

- Print out the total amount and interest made over the 5 years (Program Body)

Python Definition Solution 3

- Modify the below definition, to return the compound interest to the program body:

```
def compound_interest(principal_amount, interest_rate, years=1):
    amount = principal_amount * ((1 + (interest_rate / 100)) ** years)
    amount = round(amount, 2)
    return amount
```

```
prinAmount = 1000

totalAmount = mD.compound_interest(prinAmount, 3.23, 5)

print("Your total amount is:", totalAmount)
print("Interest            :", totalAmount - prinAmount)
```

Python Definition Calls Definition

- We already have the definition written for volume of a sphere
- Generic to Mathematics

```
def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    return volume
```

- If we were to create a definition for mass of a planet we would be duplicating code.

```
def mass_of_a_planet(radius, density):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    mass = volume * density
    return mass
```

Python Definition Calls Definition

- Definitions can call other definitions:

```
def mass_of_a_planet(radius, density):
    mass = volume_of_a_sphere(radius) * density
    return mass


def volume_of_a_sphere(radius):
    volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
    return volume
```

Python Definition Calls Definition

```
1) def mass_of_a_planet(radius, density):
2)     mass = volume_of_a_sphere(radius) * density
3)     return mass


4) def volume_of_a_sphere(radius):
5)     volume = (4.0 / 3.0) * 3.14 * (radius ** 3)
6)     return volume
```

- Def called by program body 1)
- Line 2), sends radius to definition at line 4), volume calculations are completed line 5) with data returned from line 6), line 2) multiplies by density and stores as mass
- Line 3) Returns mass to program body (or other definition)